ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

Кафедра

«Криптология и кибербезопасность»

Контрольная работа №2

Исполнитель:

студент гр. Б19-515 Щербакова А. Е.

подпись,

дата

Преподаватель: Анисимов Д.В.

подпись,

дата

Москва — 2023

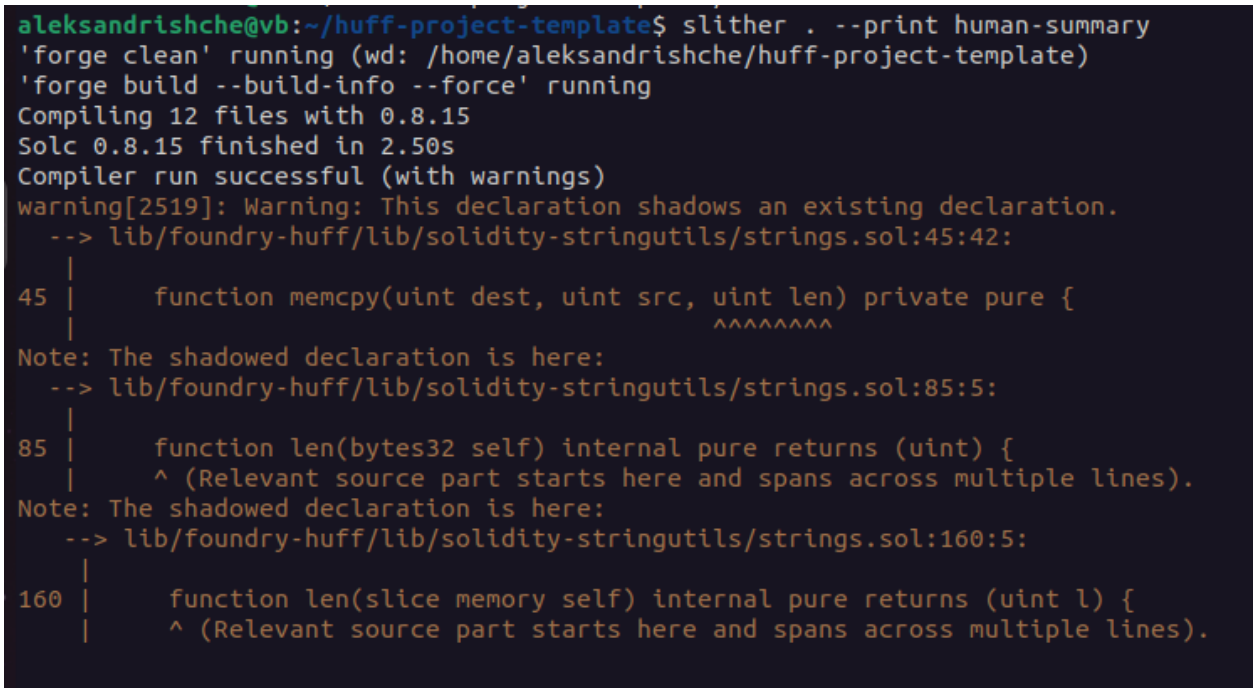
1 Описание стенда

Для анализа уязвимостей выбран следующий Github проект: https://github.com/PaulRBerg/prb-math. Написан на Solidity - языке для блокчейн-платформы Ethereum при помощи фреймфорка Foundry (для компиляции и тестирования смарт-контрактов).

В качестве статического анализатора Solidity используется Open Source проект: https://github.com/crytic/slither. В числе прочих работает и с Foundry-проектами.

Исходный код Elliptic-curve-solidity скачан с гита на виртуальной машине Ubuntu 22.04, Slither установлен на ней же при помощи pip3. Предварительно установлен Foundry: https://github.com/foundry-rs/foundry

2 Краткий отчет анализатора



```
INFO:Printers:
Compiled with Foundry
Number of lines: 16 (+ 6165 in dependencies, + 29 in tests)
Number of assembly lines: 0
Number of contracts: 0 (+ 13 in dependencies, + 4 tests)

Number of optimization issues: 2
Number of informational issues: 123
Number of low issues: 7
Number of medium issues: 13
Number of high issues: 1

+-------------+-------------+------+------------+--------------+----------+
|     Name    | # functions | ERCS | ERC20 info | Complex code | Features |
+-------------+-------------+------+------------+--------------+----------+
| SimpleStore |      2      |      |            |      No      |          |
|    Deploy   |      6      |      |            |      No      |          |
+-------------+-------------+------+------------+--------------+----------+
INFO:Slither:. analyzed (17 contracts)
```

3 Описание high уязвимостей

```
INFO:Detectors:
SimpleStore is re-used:
        - SimpleStore (script/Deploy.s.sol#7-10)
        - SimpleStore (test/SimpleStore.t.sol#26-29)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#name-reused
```

Два контракта (в данном случае интерфейса) имеют одинаковое имя, в связи с чем один из них не может быть проанализирован.

https://github.com/crytic/slither/wiki/Detector-Documentation#name-reused

Name reused

Configuration

- Check: `name-reused`
- Severity: `High`
- Confidence: `High`

Description

If a codebase has two contracts the similar names, the compilation artifacts will not contain one of the contracts with the duplicate name.

Exploit Scenario:

Bob's `truffle` codebase has two contracts named `ERC20`. When `truffle compile` runs, only one of the two contracts will generate artifacts in `build/contracts`. As a result, the second contract cannot be analyzed.

Recommendation

Rename the contract.

Проблема исправляется изменением имени одного из интерфейсов:

SimpleStore.t.sol

```
// SPDX-License-Identifier: Unlicense
pragma solidity ^0.8.15;

import "foundry-huff/HuffDeployer.sol";
import "forge-std/Test.sol";
import "forge-std/console.sol";

contract SimpleStoreTest is Test {
    /// @dev Address of the SimpleStore contract.
    SimpleStore2 public simpleStore;

    /// @dev Setup the testing environment.
    function setUp() public {
        simpleStore = SimpleStore2(HuffDeployer.deploy("SimpleStore"));
    }

    /// @dev Ensure that you can set and get the value.
    function testSetAndGetValue(uint256 value) public {
        simpleStore.setValue(value);
        console.log(value);
        console.log(simpleStore.getValue());
        assertEq(value, simpleStore.getValue());
    }
}

interface SimpleStore2 {
    function setValue(uint256) external;
    function getValue() external returns (uint256);
}
```

Проверено, что после этого действия уязвимость устранена. Кроме того, в контракте с измененным именем также не найдено уязвимостей.

```
Number of optimization issues: 2
Number of informational issues: 123
Number of low issues: 7
Number of medium issues: 13
Number of high issues: 0
```

4 Описание medium уязвимостей

1) **Деление перед умножением** – может привести к точности.

```
INFO:Detectors:
strings.ord(strings.slice) (lib/foundry-huff/lib/solidity-stringutils/strings.sol#301-343) performs a multiplication on the result of a division:
        - b = word / divisor (lib/foundry-huff/lib/solidity-stringutils/strings.sol#312)
        - ret = b (lib/foundry-huff/lib/solidity-stringutils/strings.sol#314)
        - ret = (ret * 64) | (b & 0x3F) (lib/foundry-huff/lib/solidity-stringutils/strings.sol#339)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```

Divide before multiply

Configuration

- Check: `divide-before-multiply`
- Severity: `Medium`
- Confidence: `Medium`

Description

Solidity's integer division truncates. Thus, performing division before multiplication can lead to precision loss.

Exploit Scenario:

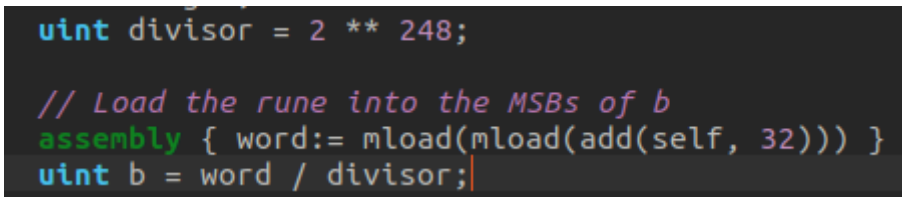
```
contract A {
        function f(uint n) public {
        coins = (oldSupply / n) * interest;
    }
}
```

If `n` is greater than `oldSupply`, `coins` will be zero. For example, with `oldSupply = 5; n = 10, interest = 2`, coins will be zero.
If `(oldSupply * interest / n)` was used, `coins` would have been `1`.
In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

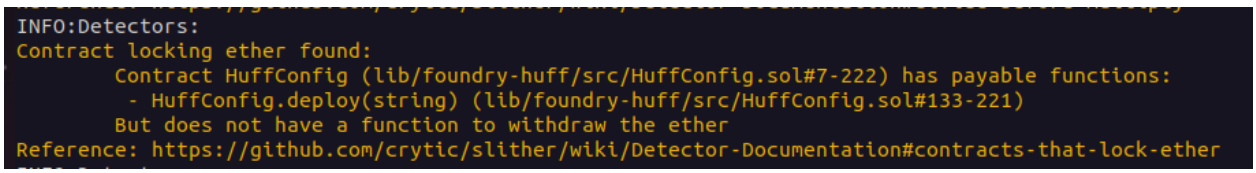
Recommendation

Consider ordering multiplication before division.

Если прочитать код, то видно, что это не уязвимость, а целенаправленное действие: число делится на 2^248, чтобы откинуть остаток. Исправлять ничего не требуется.



2) **Контракт с функцией оплаты, но без возможности вывода средств** – грозит потерей Эфира.



Contracts that lock Ether

Configuration

- Check: `locked-ether`
- Severity: `Medium`
- Confidence: `High`

Description

Contract with a `payable` function, but without a withdrawal capacity.

Exploit Scenario:

```
pragma solidity 0.4.24;
contract Locked{
    function receive() payable public{
    }
}
```

Every Ether sent to `Locked` will be lost.

Recommendation

Remove the payable attribute or add a withdraw function.

Лечится удалением модификатора *payable* или добавлением действия *owner.transfer(ether)* в зависимости от назначения контракта.

3) Далее идут 8 однотипных уязвимостей: **локальная переменная не инициализирована**.

```
INFO:Detectors:
Test.rawToConvertedReceipts(Test.RawReceipt[]).i (lib/forge-std/src/Test.sol#763) is a local variable never initialized
Test.rawToConvertedEIPTx1559s(Test.RawTx1559[]).i (lib/forge-std/src/Test.sol#674) is a local variable never initialized
strings.len(bytes32).ret (lib/foundry-huff/lib/solidity-stringutils/strings.sol#86) is a local variable never initialized
Test.readEIP1559ScriptArtifact(string).artifact (lib/forge-std/src/Test.sol#658) is a local variable never initialized
Test.rawToConvertedReceipt(Test.RawReceipt).receipt (lib/forge-std/src/Test.sol#773) is a local variable never initialized
Test.rawToConvertedEIPTx1559(Test.RawTx1559).transaction (lib/forge-std/src/Test.sol#684) is a local variable never initialized
Test.rawToConvertedEIP1559Detail(Test.RawTx1559Detail).txDetail (lib/forge-std/src/Test.sol#698) is a local variable never initialized
Test.rawToConvertedReceiptLogs(Test.RawReceiptLog[]).i (lib/forge-std/src/Test.sol#795) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
```

Uninitialized local variables

Configuration

- Check: `uninitialized-local`
- Severity: `Medium`
- Confidence: `Medium`

Description

Uninitialized local variables.

Exploit Scenario:

```
contract Uninitialized is Owner{
    function withdraw() payable public onlyOwner{
        address to;
        to.transfer(this.balance)
    }
}
```

Bob calls `transfer`. As a result, all Ether is sent to the address `0x0` and is lost.

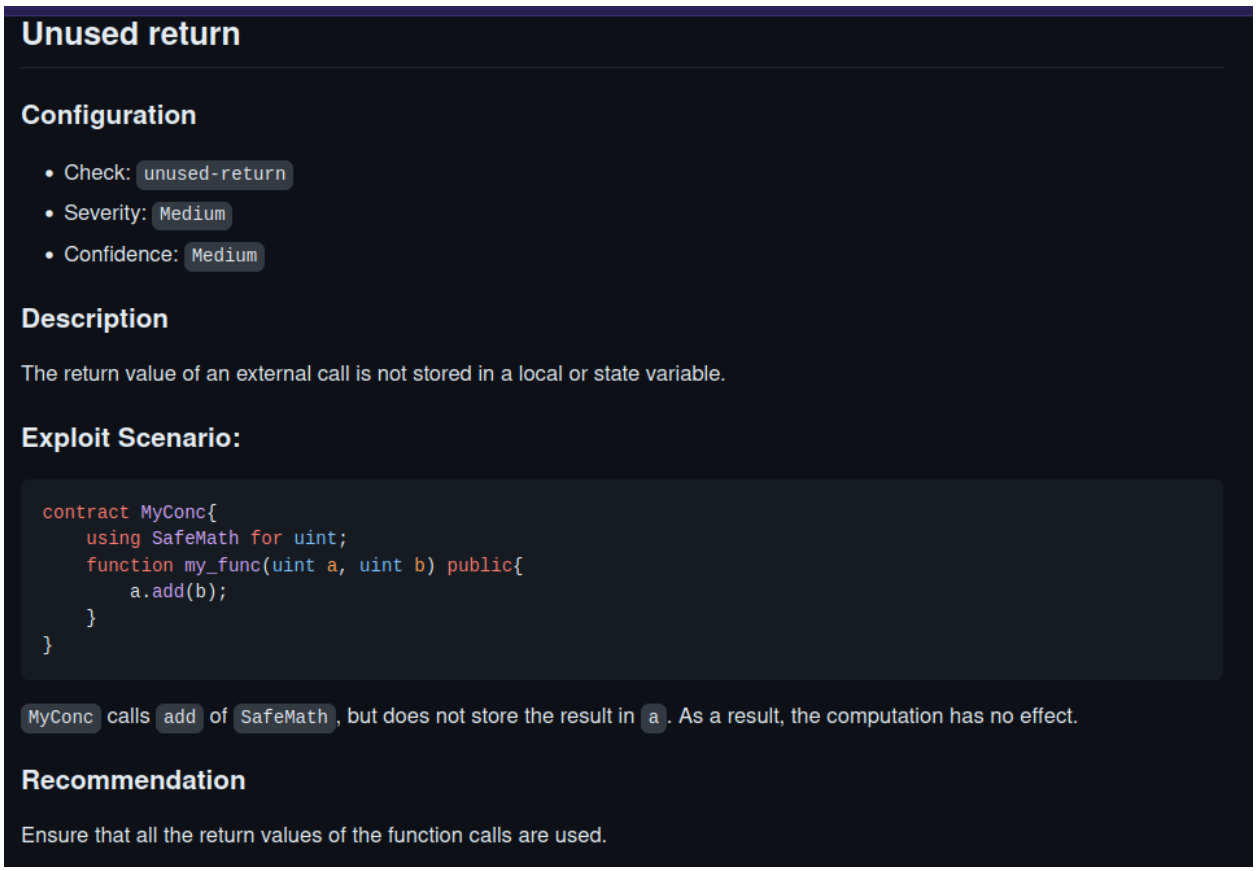
Recommendation

Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

Решение: удалить переменную, если она не используется или присвоить ей значение, если используется.

4) **Неиспользуемое возвращаемое значение** – грозит избыточными вычислениями без результата. 3 однотипные ошибки.

```
INFO:Detectors:
HuffConfig.deploy(string) (lib/foundry-huff/src/HuffConfig.sol#133-221) ignores return value by vm.ffi(create_cmds) (lib/foundry-huff/src/HuffConfig.sol#170)
HuffConfig.deploy(string) (lib/foundry-huff/src/HuffConfig.sol#133-221) ignores return value by vm.ffi(append_cmds) (lib/foundry-huff/src/HuffConfig.sol#177)
HuffConfig.deploy(string) (lib/foundry-huff/src/HuffConfig.sol#133-221) ignores return value by vm.ffi(cleanup) (lib/foundry-huff/src/HuffConfig.sol#204)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```



Unused return

Configuration

- Check: `unused-return`
- Severity: `Medium`
- Confidence: `Medium`

Description

The return value of an external call is not stored in a local or state variable.

Exploit Scenario:

```
contract MyConc{
    using SafeMath for uint;
    function my_func(uint a, uint b) public{
        a.add(b);
    }
}
```

`MyConc` calls `add` of `SafeMath`, but does not store the result in `a`. As a result, the computation has no effect.

Recommendation

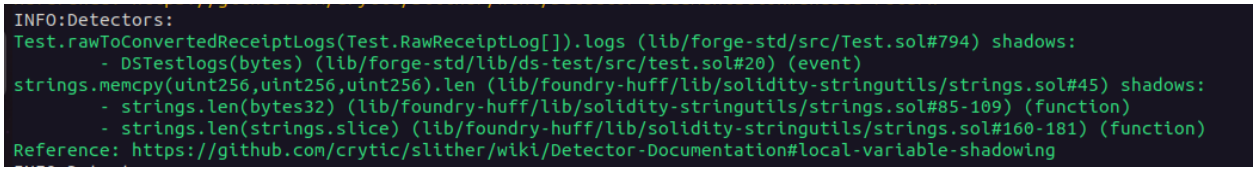
Ensure that all the return values of the function calls are used.

Все 3 выявленные уязвимости заключаются в применении функции *vm.ffi(args);* без присвоения ее результата. Эта функция компилирует контракт на Huff и возвращает байткод контракта.

Для некоторых вспомогательных контрактов не требуется запоминать байткод, так как они используются один раз, следовательно, это можно считать ложноположительным срабатыванием.

5 Описание low уязвимостей

1) **Одинаковое название локальных переменных** – 2 одинаковых уязвимости. Грозит некорректной работой.



```
INFO:Detectors:
Test.rawToConvertedReceiptLogs(Test.RawReceiptLog[]).logs (lib/forge-std/src/Test.sol#794) shadows:
        - DSTestlogs(bytes) (lib/forge-std/lib/ds-test/src/test.sol#20) (event)
strings.memcpy(uint256,uint256,uint256).len (lib/foundry-huff/lib/solidity-stringutils/strings.sol#45) shadows:
        - strings.len(bytes32) (lib/foundry-huff/lib/solidity-stringutils/strings.sol#85-109) (function)
        - strings.len(strings.slice) (lib/foundry-huff/lib/solidity-stringutils/strings.sol#160-181) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
```

Local variable shadowing

Configuration

- Check: `shadowing-local`
- Severity: `Low`
- Confidence: `High`

Description

Detection of shadowing using local variables.

Exploit Scenario:

```
pragma solidity ^0.4.24;

contract Bug {
    uint owner;

    function sensitive_function(address owner) public {
        // ...
        require(owner == msg.sender);
    }

    function alternate_sensitive_function() public {
        address owner = msg.sender;
        // ...
        require(owner == msg.sender);
    }
}
```

`sensitive_function.owner` shadows `Bug.owner`. As a result, the use of `owner` in `sensitive_function` might be incorrect.

Recommendation

Rename the local variables that shadow another component.

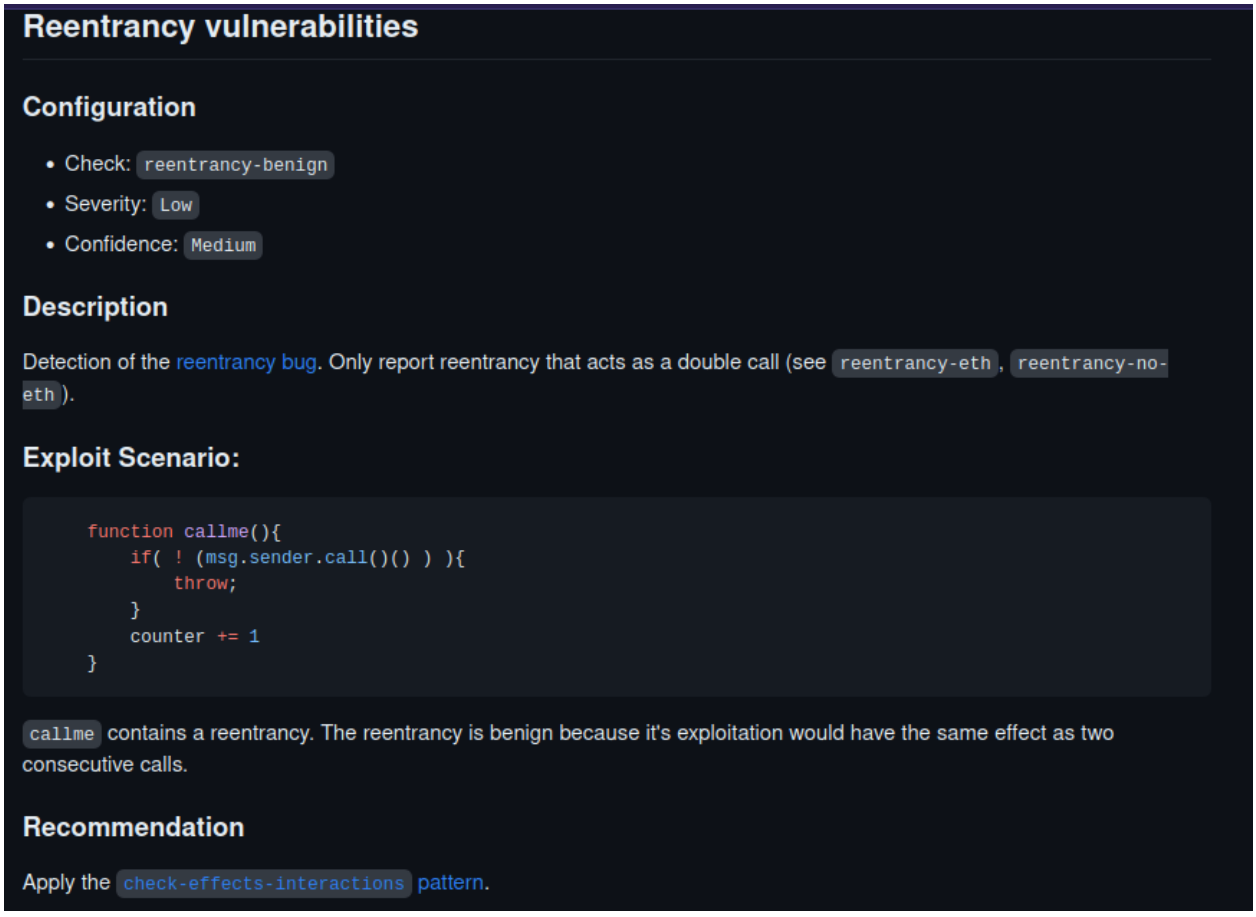
Первая уязвимость: устранена переменованием переменной logs на logs2.


```
function rawToConvertedReceiptLogs(RawReceiptLog[] memory rawLogs)
    internal pure
    returns (ReceiptLog[] memory)
{
    ReceiptLog[] memory logs2 = new ReceiptLog[](rawLogs.length);
    for (uint i; i < rawLogs.length; i++) {
        logs2[i].logAddress = rawLogs[i].logAddress;
        logs2[i].blockHash = rawLogs[i].blockHash;
        logs2[i].blockNumber = bytesToUint(rawLogs[i].blockNumber);
        logs2[i].data = rawLogs[i].data;
        logs2[i].logIndex = bytesToUint(rawLogs[i].logIndex);
        logs2[i].topics = rawLogs[i].topics;
        logs2[i].transactionIndex = bytesToUint(rawLogs[i].transactionIndex);
        logs2[i].transactionLogIndex = bytesToUint(rawLogs[i].transactionLogIndex);
        logs2[i].removed = rawLogs[i].removed;
    }
    return logs2;

}
```

Вторая уязвимость: в одну функцию передается параметр, одноименный другой функции. Исправлено переменованием len на len2.

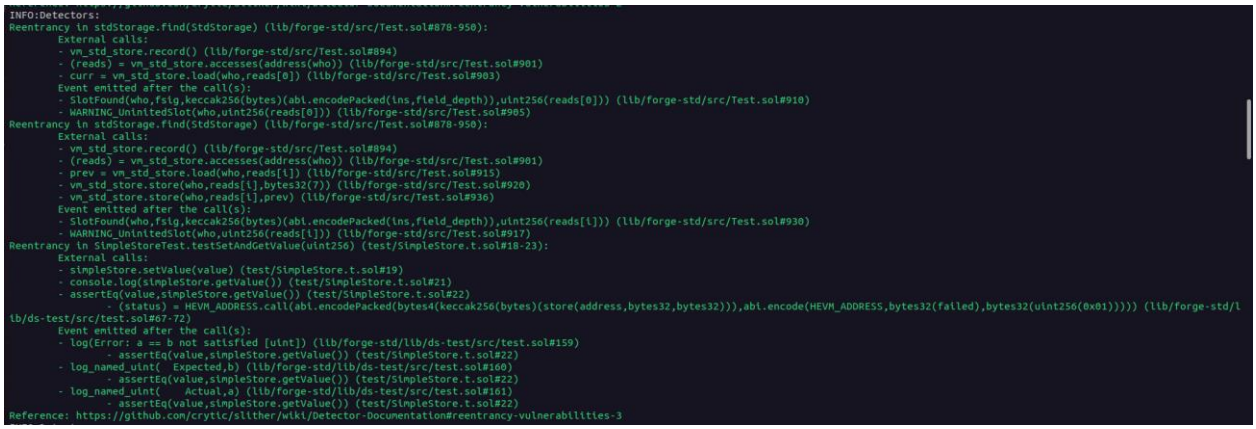
```
function memcpy(uint dest, uint src, uint len2) private pure {
    // Copy word-length chunks while possible
    for(; len2 >= 32; len2 -= 32) {
        assembly {
            mstore(dest, mload(src))
        }
        dest += 32;
        src += 32;
    }

    // Copy remaining bytes
    uint mask = type(uint).max;
    if (len2 > 0) {
        mask = 256 ** (32 - len2) - 1;
    }
    assembly {
        let srcpart := and(mload(src), not(mask))
        let destpart := and(mload(dest), mask)
        mstore(dest, or(destpart, srcpart))
    }
}
```

2) **Повторный вход** – 2 уязвимости.

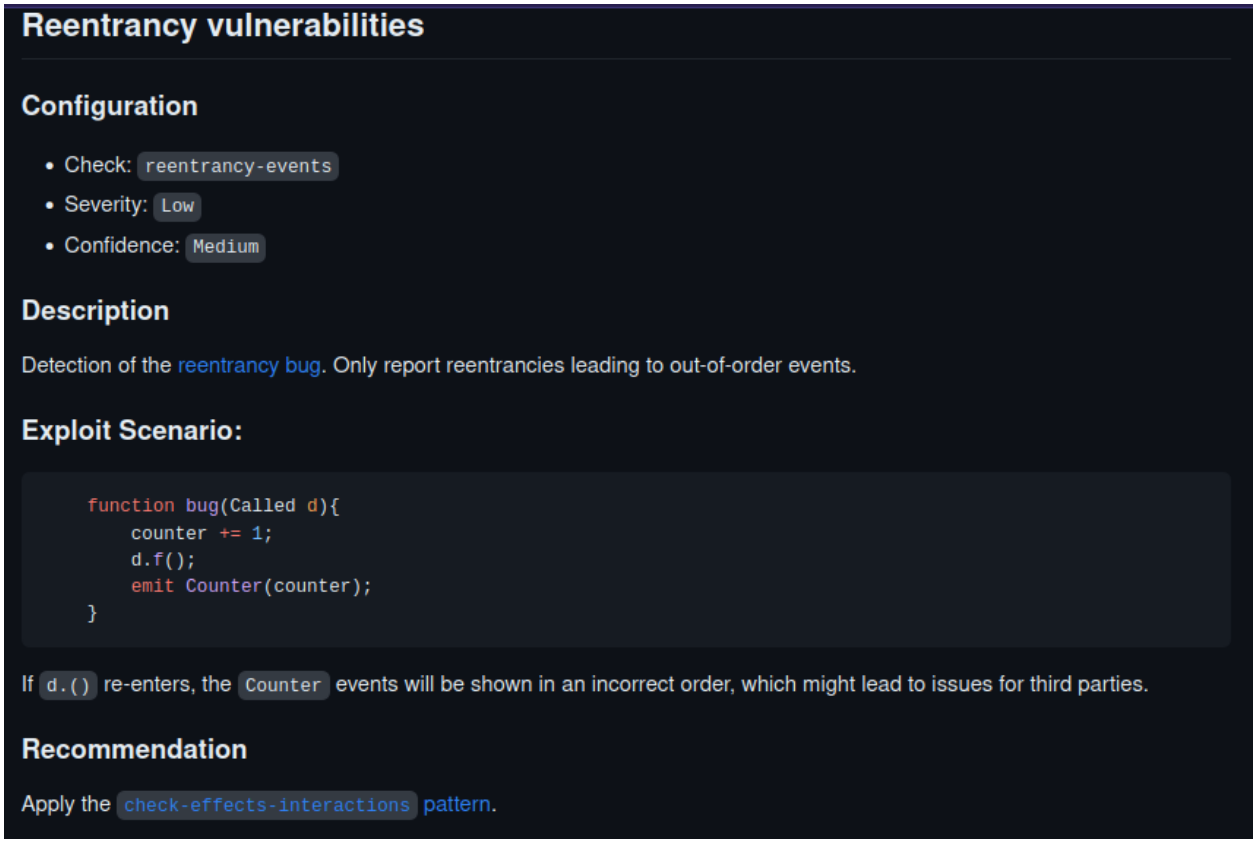


Это «доброкачественная» уязвимость повторного входа (то есть она не приводит к ошибкам), так что исправление не требуется.

3) **Повторный вход** – 3 уязвимости.



```
INFO:Detectors:
Reentrancy in stdStorage.find(StdStorage) (lib/forge-std/src/Test.sol#878-950):
        External calls:
        - vm_std_store.record() (lib/forge-std/src/Test.sol#894)
        - (reads) = vm_std_store.accesses(address(who)) (lib/forge-std/src/Test.sol#901)
        - curr = vm_std_store.load(who,reads[0]) (lib/forge-std/src/Test.sol#903)
        Event emitted after the call(s):
        - SlotFound(who,fsig,keccak256(bytes)(abi.encodePacked(ins,field_depth)),uint256(reads[0])) (lib/forge-std/src/Test.sol#910)
        - WARNING_UninitedSlot(who,uint256(reads[0])) (lib/forge-std/src/Test.sol#905)
Reentrancy in stdStorage.find(StdStorage) (lib/forge-std/src/Test.sol#878-950):
        External calls:
        - vm_std_store.record() (lib/forge-std/src/Test.sol#894)
        - (reads) = vm_std_store.accesses(address(who)) (lib/forge-std/src/Test.sol#901)
        - prev = vm_std_store.load(who,reads[i]) (lib/forge-std/src/Test.sol#915)
        - vm_std_store.store(who,reads[i],bytes32(7)) (lib/forge-std/src/Test.sol#920)
        - vm_std_store.store(who,reads[i],prev) (lib/forge-std/src/Test.sol#936)
        Event emitted after the call(s):
        - SlotFound(who,fsig,keccak256(bytes)(abi.encodePacked(ins,field_depth)),uint256(reads[i])) (lib/forge-std/src/Test.sol#930)
        - WARNING_UninitedSlot(who,uint256(reads[i])) (lib/forge-std/src/Test.sol#917)
Reentrancy in SimpleStoreTest.testSetAndGetValue(uint256) (test/SimpleStore.t.sol#18-23):
        External calls:
        - simpleStore.setValue(value) (test/SimpleStore.t.sol#19)
        - console.log(simpleStore.getValue()) (test/SimpleStore.t.sol#21)
        - assertEq(value,simpleStore.getValue()) (test/SimpleStore.t.sol#22)
                - (status) = HEVM_ADDRESS.call(abi.encodePacked(bytes4(keccak256(bytes)(store(address,bytes32,bytes32))),abi.encode(HEVM_ADDRESS,bytes32(failed),bytes32(uint256(0x01))))) (lib/forge-std/l
ib/ds-test/src/test.sol#67-72)
        Event emitted after the call(s):
        - log(Error: a == b not satisfied [uint]) (lib/forge-std/lib/ds-test/src/test.sol#159)
                - assertEq(value,simpleStore.getValue()) (test/SimpleStore.t.sol#22)
        - log_named_uint(  Expected,b) (lib/forge-std/lib/ds-test/src/test.sol#160)
                - assertEq(value,simpleStore.getValue()) (test/SimpleStore.t.sol#22)
        - log_named_uint(    Actual,a) (lib/forge-std/lib/ds-test/src/test.sol#161)
                - assertEq(value,simpleStore.getValue()) (test/SimpleStore.t.sol#22)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```



Reentrancy vulnerabilities

Configuration

- Check: `reentrancy-events`
- Severity: `Low`
- Confidence: `Medium`

Description

Detection of the reentrancy bug. Only report reentrancies leading to out-of-order events.

Exploit Scenario:

```
function bug(Called d){
    counter += 1;
    d.f();
    emit Counter(counter);
}
```

If `d.()` re-enters, the `Counter` events will be shown in an incorrect order, which might lead to issues for third parties.

Recommendation

Apply the `check-effects-interactions` pattern.

«Злокачественная» уязвимость повторного входа, которая может привести к ошибкам. Требуется исправление:

```
pragma solidity ^0.4.0;

// THIS CONTRACT CONTAINS A BUG - DO NOT USE
contract Fund {
    /// Mapping of ether shares of the contract.
    mapping(address => uint) shares;
    /// Withdraw your share.
    function withdraw() public {
        if (msg.sender.call.value(shares[msg.sender])())
            shares[msg.sender] = 0;
    }
}
```

To avoid re-entrancy, you can use the Checks-Effects-Interactions pattern as outlined further below:

```
pragma solidity ^0.4.11;

contract Fund {
    /// Mapping of ether shares of the contract.
    mapping(address => uint) shares;
    /// Withdraw your share.
    function withdraw() public {
        var share = shares[msg.sender];
        shares[msg.sender] = 0;
        msg.sender.transfer(share);
    }
}
```